Нижегородский государственный университет им. Н.И.Лобачевского

Факультет Вычислительной математики и кибернетики

Операционные системы: *аспекты параллелизма*

Синхронизация-1

Линёв А.В.

2007

Тема обсуждения

- При переходе от последовательных решений к параллельным возникает проблема синхронизации
  - обеспечение согласованных действий параллельно работающих модулей при выполнении этапов алгоритма
  - обеспечение целостности используемых данных
  - извещение модулями друг друга о произошедших событиях

  и т.д.

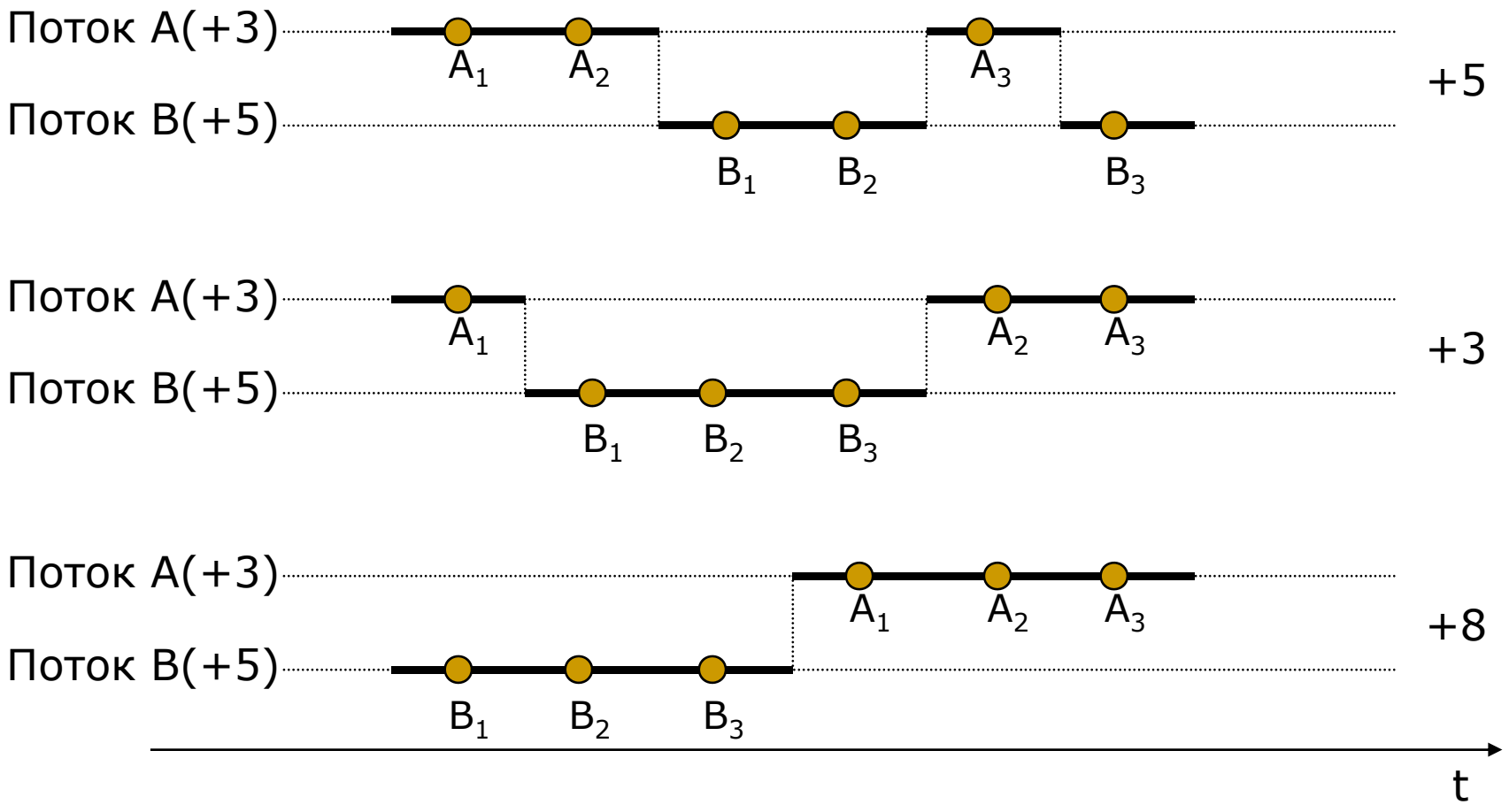
Необходимость синхронизации

Пример многопоточного приложения

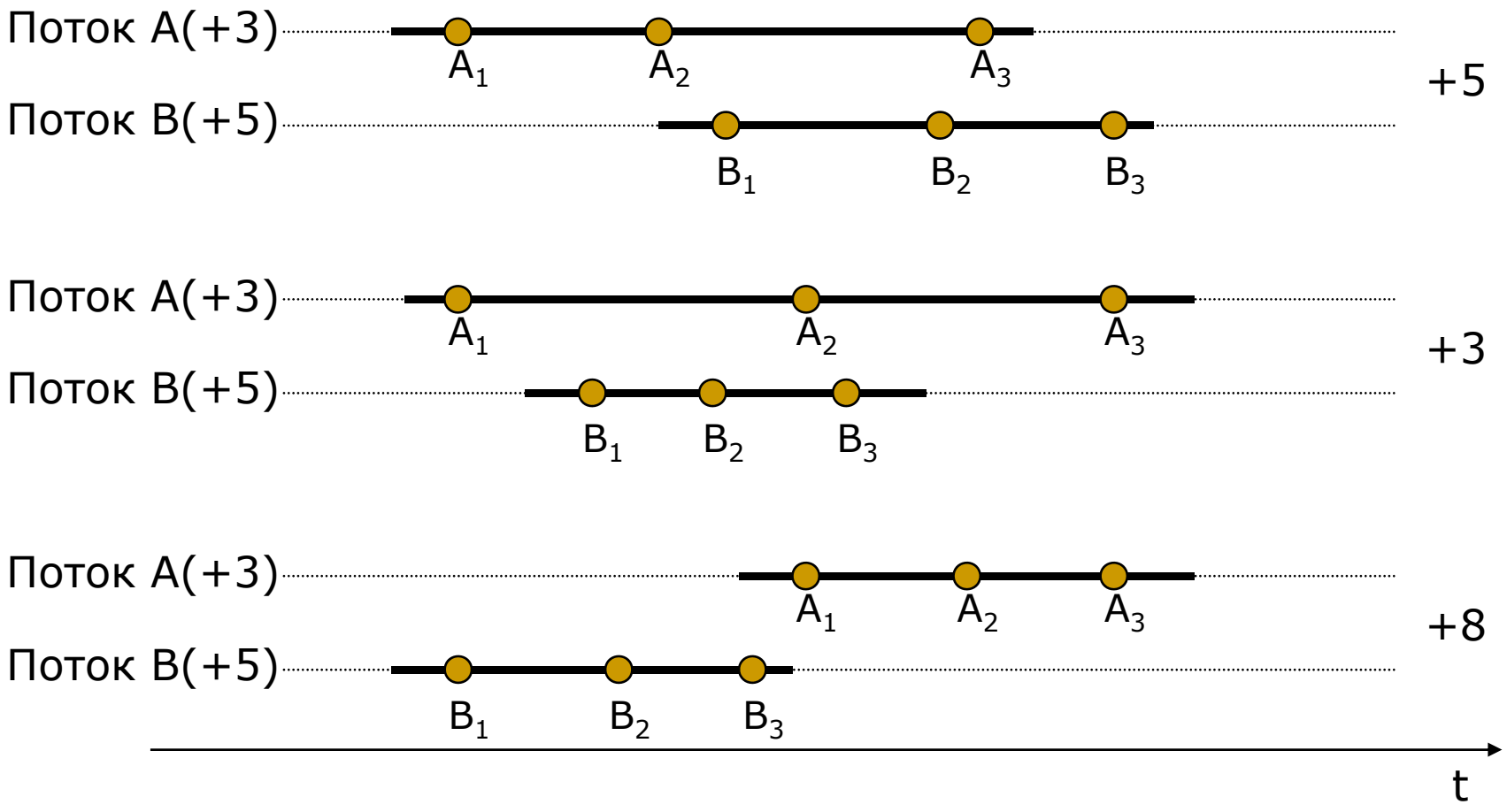
Пусть есть 2 потока, совместно работающих с общей базой данных. При работе они используют однотипные алгоритмы, включающие следующие три шага:

1. Считать из базы данных в локальный буфер потока запись о клиенте с заданным идентификатором
2. Изменить значение некоторого поля
3. Записать модифицированную запись в базу данных

Диаграммы выполнения потоков на однопроцессорной системе



Диаграммы выполнения потоков на многопроцессорной системе



Проблема синхронизации…

- Результат вычислений в рассмотренном примере не однозначен – все зависит от условий выполнения потоков (разные запуски программы могут привести к разным результатам !!!)
- Точно такие ситуации могут возникать и при использовании "обычных" общих переменных потоков

Проблема синхронизации

- Сложность проблемы синхронизации кроется в нерегулярности возникающих ситуаций – все определяется взаимными скоростями потоков и моментами передачи ЦП от одного потока к другому
- Ситуация, когда два или более потоков обрабатывают разделяемые данные и конечный результат зависит от соотношения скоростей потоков, называется **состязанием** или **гонками** (*race conditions*)

Критическая секция…

- **Критическая секция (КС, *critical section*)** – это часть программы, результат выполнения которой может непредсказуемо меняться, если в ходе ее выполнения состояние ресурсов, используемых в этой части программы, изменяется другими потоками
- Критическая секция всегда определяется по отношению к определенным **критическим ресурсам** (например, **критическим данным**), при несогласованном доступе к которым могут возникнуть нежелательные эффекты

Критическая секция

- Чтобы исключить эффект гонок по отношению к критическим данным, необходимо обеспечить, чтобы в каждый момент времени в критической секции, связанной с этими данными, находился только один поток. Все остальные потоки должны блокироваться на входе в критическую секцию. Подобное требование обычно называется **взаимоисключением (*mutual exclusion*)**
- Когда один поток покидает критическую секцию, один из ожидающих потоков может в нее войти

Задача взаимного исключения

Постановка задачи

Задача взаимного исключения. Постановка задачи

- Необходимо согласовать работу n>1 параллельных потоков при использовании некоторого критического ресурса таким образом, чтобы удовлетворить следующим требованиям
  - одновременно внутри критической секции должно находиться не более одного потока
  - критические секции не должны иметь приоритета в отношении друг друга
  - остановка какого-либо потока вне его критической секции не должна влиять на дальнейшую работу потоков по использованию критического ресурса
  - решение о вхождении потоков в их критические секции при одинаковом времени поступления запросов на такое вхождение и равноприоритетности потоков не откладывается на неопределенный срок, а является конечным во времени
  - относительные скорости развития потоков неизвестны и произвольны
  - любой поток может переходить в любое состояние, отличное от активного, вне пределов своей критической секции
  - освобождение критического ресурса и выход из критической секции должны быть произведены потоком, использующим критический ресурс, за конечное время

Задача взаимного исключения

Аппаратная поддержка решения

Задача взаимного исключения. Аппаратная поддержка решения

- Запрещение прерываний
- Использование разделяемых переменных
  - Алгоритм Деккера
  - Алгоритм Петерсона
  - Алгоритм булочной
- Использование специальных команд ЦП
  - test-and-set
  - swap
  - read-conditional-write

Запрещение прерываний

```
while( true ){
   DisableInterrupts();
   CS();       /*   Critical Section   */
   EnableInterrupts();
   NCS();          /* Non-Critical Section */
}
```

- Прикладные программы, как правило, не могут запрещать прерывания
- На многопроцессорных системах прерывания запрещаются только на текущем процессоре, соответственно, несколько ЦП одновременно могут выполнять критическую секцию
- При запрещении прерываний на длительное время могут возникнуть сложности с функционированием устройств, не получавших должного внимания

Использование разделяемых переменных – способ №1

```
bool flag = false;
```

```
while( true ){
1  while( flag ) ;
5  flag = true;
6  CS1();
   flag = false;
   NCS1();
}
```

```
while( true ){
2  while( flag ) ;
3  flag = true;
4  CS2();
   flag = false;
   NCS2();
}
```

Неверное решение

Два потока могут одновременно находиться в критической секции – не удовлетворяется требование взаимоисключения

Использование разделяемых переменных – способ №2

```
bool flag1 = false;
bool flag2 = true;
while( true ){
1  while( flag2 ) ;
5  flag1 = true;
6  CS1();
   flag1 = false;
   NCS1();
}
```

```
while( true ){
2  while( flag1 ) ;
3  flag2 = true;
4  CS2();
   flag2 = false;
   NCS2();
}
```

Неверное решение

Два потока опять могут одновременно находиться в критической секции – снова не удовлетворяется требование взаимоисключения

Использование разделяемых переменных – способ №3

```
bool flag1 = false;
bool flag2 = false;
while( true ){
1 flag1 = true;
4 while( flag2 ) ;
  CS1();
  flag1 = false;
  NCS1();
}
```

```
while( true ){
2 flag2 = true;
3 while( flag1 ) ;
  CS2();
  flag2 = false;
  NCS2();
}
```

Неверное решение

Потоки могут попасть в бесконечный цикл – не выполняется условие конечности времени принятии решения о входе в КС

Использование разделяемых переменных – алгоритм Деккера

```
/* Номер потока: 1 или 2. Приведен код потока 1. */
flag1 = false; flag2 = false; turn = 1;
/* flag_i – намерение i-го потока войти в КС */
/* turn – номер потока, имеющего право войти первым */
while( true ){
   flag1 = true;
l1:if( flag2 ){
      if( turn == 1 )
         goto l1;
      else{
         flag1 = false;
         while( turn == 2 ){}
      }
   }
   else{
      CS_i(); turn = 2; flag1 = false;
   }
}
```

Использование разделяемых переменных – алгоритм Петерсона

```
/* i – номер потока (0 или 1) */

int ready[2] = {0, 0};  /* Намерение войти в КС */
int turn = 0;           /* Приоритетный поток */

while( true ) {
   ready[i] = 1;
   turn = 1 - i;
   while( ready[1-i] && turn == 1-i )
      ;
   CS_i();
   ready[i] = 0;
   NCS_i();
}
```

Использование разделяемых переменных – алгоритм булочной…

- В алгоритме булочной (*bakery algorithm*) организуется очередь из потоков, желающих войти в критическую секцию. Для этого используются следующие разделяемые переменные:
  - int **number**[n] = { 0, 0, … } – i-ый элемент массива хранит позицию i-ого потока в очереди в критическую секцию (0 означает, что поток не пытается попасть в КС) – если два потока имеют одинаковую позицию в очереди, первым в критическую секцию попадает поток, имеющий меньший номер
  - enum {false, true} **choosing**[n] = {false,false,…} – i-ый элемент массива равен true во время вычисления i-ым потоком своей позиции в очереди

Использование разделяемых переменных – алгоритм булочной

```
enum {false, true} choosing[n] = { false, false, ... };
int number[n] = { 0, 0, … };
while( true ) {
  choosing[i] = true;
  number[i] = max(number[0], ..., number[n-1]) + 1;
  choosing[i] = false;
  for( j = 0; j < n; j++ ){
    while(choosing[j])
      ;
    while(number[j] != 0 && (number[j],j) < (number[i],i))
      ;
  }
  CS_i();
  number[i] = 0;
  NCS_i();
}
```

Использование специальных команд ЦП…

- Команда Test-and-Set (Проверить и присвоить 1)

```
int Test_and_Set (int *target){
    int tmp = *target;
    *target = 1;
    return tmp;
}
```

- Решение задачи взаимного исключения

```
int lock = 0; /* Признак блокировки критических данных */
while( true ) {
    while( Test_and_Set( &lock ) )
        ;
    CS_i();
    lock = 0;
    NCS_i();
}
```

Использование специальных команд ЦП…

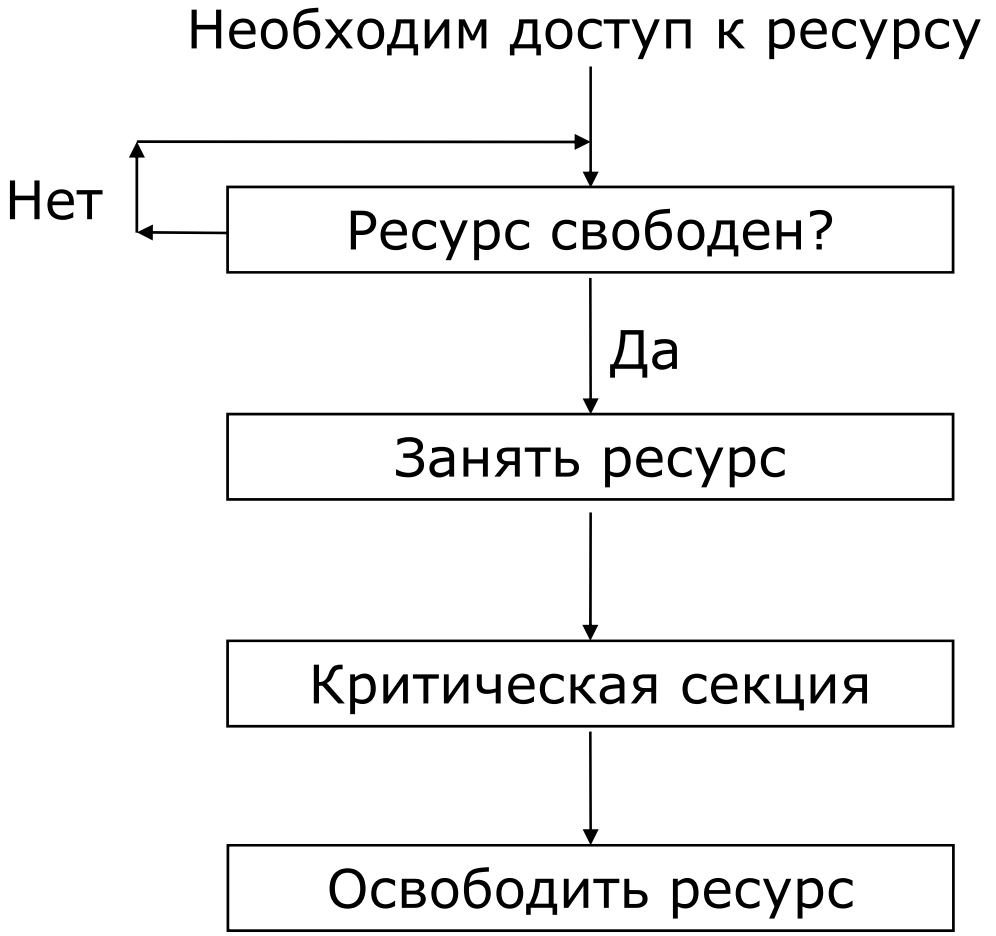
- Команда Swap (Обменять значения)

```
void Swap( int *a, int *b ){
   int tmp = *a;
   *a = *b;
   *b = tmp;
}
```

- Решение задачи взаимного исключения

```
int lock = 0; /* Признак блокировки критических данных */
int key;
while( true ) {
   key = 1;
   do{
      Swap( &lock, &key );
   }while (key);
   CS_i();
   lock = 0;
   NCS_i();
}
```

Использование специальных команд ЦП…



При работе с **признаком занятости** (**блокировки**) ресурса поток использует "активное ожидания", то есть крутится в цикле, ожидая освобождения признака блокировки

Использование специальных команд ЦП

- Недостатки активного ожидания
  - Активное ожидание работает, но оно очень затратно: если какой-то поток находится в состоянии активного ожидания, поток, захвативший признак блокировки, не может продолжать свое исполнение и никакой другой поток тоже не может
  - Активное ожидание используется как база для построения высокоуровневых механизмов
  - Существуют ситуации, когда активное ожидание оправдано

Заключение

- Понимание реализации параллельного выполнения потоков – ключевой момент при разработке и отладке параллельных программ
- При разработке параллельных программ необходимо решать задачу взаимного исключения
  - Можно самостоятельно реализовать низкоуровневый механизм, используя один из существующих алгоритмов
  - Можно использовать механизмы синхронизации, предоставляемые операционной системой

Вопросы для обсуждения

- В каких случаях оправдано использование активного ожидания?
- Будет ли алгоритм Петерсона работать на многопроцессорной системе?

Задания для самостоятельной работы

- Напишите многопоточную программу, содержащую гонки и не обеспечивающую взаимное исключение. Подберите параметры таким образом, чтобы более чем в 99% случаев программа получала правильный результат.

Литература

1. Таненбаум Э. Современные операционные системы. 2-е изд. - СПб.: Питер, 2002.
2. Карпов В.Е., Коньков К.А. Введение в операционные системы. Курс лекций. 2-е изд. - М.: ИНТУИТ.РУ, 2005.

Тема следующей лекции – Синхронизация-2

- Высокоуровневые механизмы синхронизации:
  - Семафоры
  - Мониторы